

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

5

**APPLICATION PAPERS**

10

**OF**

15

**TODD MICHAEL AUSTIN**

**DAVID THEODORE BLAAUW**

**TREVOR NIGEL MUDGE**

20

**KRISZTIAN FLAUTNER**

25

**FOR**

**DATA RETENTION LATCH PROVISION**  
**WITHIN INTEGRATED CIRCUITS**

## **BACKGROUND OF THE INVENTION**

### **Field of the Invention**

5           This invention relates to the field of data processing systems. More particularly, this invention relates to the field of integrated circuits having data retention latches used to retain the state of the integrated circuit in a standby mode, during which the integrated circuit does not perform processing operations.

### **Description of the Prior Art**

15           It is known to provide data retention latches (sometimes referred to as balloon latches) within integrated circuits for the purpose of retaining state data in a low power standby mode of operation. These data retention latches can be formed to have electrical characteristics well suited to their low power standby mode type of operation, e.g. they can have a high threshold voltage resulting in low leakage current losses. Such low power characteristics are normally not suitable for use in the operational circuits which must switch at high speed during data processing. However, the data retention latches need not operate at high speed and so can be engineered with low power consumption in mind. When entering the standby mode, the state data is typically transferred from the operational latch into the data retention latch and then the operational latch is powered down. On exiting the standby mode, the stored data value held within the data retention latch is forced back from the data retention latch into the operational latch and processing operations are recommenced. 20           This provides advantageously rapid entry and exit from standby mode without data loss.

30           A problem associated with the use of data retention latches in this way is the large number of data retention latches which typically need to be provided on an integrated circuit to hold sufficient of its state that processing operations can be stopped and restarted without data loss. The circuit area consumed by such a large number of data retention latches is a disadvantageous overhead.

It has been proposed that in order to achieve better performance, in terms of speed and/or energy consumption, operational latches within an integrated circuit should be associated with delayed latches serving to capture the same signal value as the operational latch but at a slightly later time. The two captured values can then be compared and if the delayed captured value differs from the earlier captured value, then corrective action may be taken. Such an arrangement allows the circuit to be operated closer to the limiting levels of speed and/or voltage etc. Such an arrangement will typically control operational parameters such that a finite error rate is maintained in a recognition that the penalty associated with recovering from such errors is more than outweighed by the advantages achieved in running closer to the operational limits of the integrated circuit.

### **SUMMARY OF THE INVENTION**

Viewed from one aspect the present invention provides an integrated circuit comprising:

a plurality of processing stages, at least one of said processing stages having processing logic operable to perform a processing operation upon at least one processing stage input value to generate a processing logic output signal; and

a low power mode controller operable to control said integrated circuit to switch between an operational mode in which said integrated circuit performs said processing operations and a standby mode in which said integrated circuit retains signal values but does not perform said processing operations; wherein

said at least one of said processing stages has:

a non-delayed latch operable to capture a non-delayed value of said processing logic output signal at a non-delayed capture time; and

a delayed latch operable during said operational mode to capture a delayed value of said processing logic output signal at a delayed capture time, said delayed capture time being later than said non-delayed capture time, said non-delayed value being passed as a processing stage input value to a following processing stage before said delayed capture time and a difference between said non-delayed value and said delayed value being indicative of said processing operation not being complete at said non-delayed capture time;

said delayed latch is operable during said standby mode to retain said delayed value whilst said non-delayed latch is powered down and is susceptible to loss of said non-delayed value; and

said delayed latch is formed to have a lower static power consumption.

5

The invention recognises that the delayed latches which are provided in accordance with the above-described technique may be advantageously reused as data retention latches in a standby mode. The delayed latches need not be as quick in their operation as the non-delayed latches, since their function is to capture a signal value at  
10 a later time and so they are able to operate with fewer speed constraints. Thus, the delayed latches may be formed to have the desirable lower power consumption characteristics of data retention latches and still fulfil their role as delayed latches. There is a surprising synergy between the provision of data retention latches and delayed latches in accordance with the above technique, which allows an  
15 advantageous reduction in gate count/circuit area by the double-use of the latches, whilst substantially avoiding any need to compromise in the ability of such latches to perform either of their roles.

Whilst it will be appreciated that the delayed latch is formed to have a lower  
20 power consumption and this can be achieved without comprising the circuit's overall speed of operation, in many embodiments a reduced speed of operation of the delayed latch is a preferred compromise to make in order to facilitate lower power consumption.

25 When restarting processing following the standby mode in preferred embodiments, the delayed value stored within the delayed latch is passed to a following processing stage as a processing stage input value. In some embodiments the delayed value is also supplied to the non-delayed latch when processing is restarted.

30

The action of the non-delayed latch and the delayed latch in providing a fault tolerant type of operation is conveniently provided by the action of a comparator operable to compare the non-delayed and the delayed value and error-repair logic

operable to suppress use of the non-delayed value if the comparison shows that the captured values are not the same.

It is preferred that the delayed value is used as a replacement for the non-delayed value at the processing stage output. This helps to ensure forward progress through processing operations even when errors occur.

The system advantageously further includes a meta-stability detector operable to detect meta-stability in the non-delayed value and to trigger the error-repair logic to suppress use of the non-delayed value if found to be meta-stable.

The delayed value may optionally be forced into the non-delayed latch in place of the non-delayed value when the comparator detects an error in the non-delayed value. This mechanism may also be used to restore a value held within the delayed latch during standby mode back into the non-delayed latch prior to restarting processing operations.

The delayed latch and the non-delayed latch are preferably driven by respective clock signals having a controlled phase difference therebetween in order to provide the delayed time.

It will be appreciated that the processing stages may be respective pipeline stages within a synchronised pipeline and the present technique is well suited to such operations. It will also be appreciated that the processing stages may not conduct processing operations themselves and a processing stage may correspond to a simple connection/bus connection between latches of a pipelined bus along which a signal value must pass. It will be further appreciated that the term latch should be interpreted broadly to encompass any circuit element for storing a data value irrespective of its particular timing and triggering needs.

In order to simplify the interaction between the delayed latch and its pipeline stage, it is preferred that the minimum processing time taken for the processing operation of that pipeline stage is greater than the time separating the delayed capture time from the non-delayed capture time such that the delayed value is not influenced

by a processing operation performed upon different input values than those which resulted in the non-delayed value.

In order to achieve an appropriate delay, the processing logic may desirably include one or more delay elements to ensure a minimum processing time within the processing stage is suitably exceeded.

Whilst the present technique could be used in a variety of different situations, it is particularly well suited for use as part of a data processor.

The error rate which is dealt with by the non-delayed latch and the delayed latch is desirably controlled to correspond to a finite non-zero level, whereby a balance is struck between the overhead associated in recovering from errors and the performance gains from being able to operate closer to the ultimate limits. The error rate may be desirably monitored with an error counting circuit, which may in turn be read by software for use in controlling the circuit operation and controlling parameters which influence error rate.

Alternative preferred embodiments, rather than relying on an error counting circuit to control parameters of the integrated circuit, comprise a performance monitoring module operable to monitor work quantities including a quantity of useful work performed in progressing the processing operation and a quantity of work used to perform the error-recovery operations. The integrated circuit parameters are controlled in dependence upon these work quantities. In preferred arrangements, the integrated circuit parameters include at least one of: an operating voltage; an operating frequency; an integrated circuit body-bias voltage and temperature. Viewed from another aspect the present invention provides a method of operating an integrated circuit having a plurality of processing stages, at least one of said processing stages having processing logic operable to perform a processing operation upon at least one processing stage input value to generate a processing logic output signal, said method comprising the steps of:

controlling said integrated circuit to switch between an operational mode in which said integrated circuit performs said processing operations and a standby mode

in which said integrated circuit retains signals values but does not perform said processing operations;

within said at least one of said processing stages:

capturing in a non-delayed latch a non-delayed value of said processing logic output signal at a non-delayed capture time; and

during said operational mode, capturing in a delayed latch a delayed value of said processing logic output signal at a delayed capture time, said delayed capture time being later than said non-delayed capture time, said non-delayed value being passed as a processing stage input value to a following processing stage before said delayed capture time and a difference between said non-delayed value and said delayed value being indicative of said processing operation not being complete at said non-delayed capture time; and

during said standby mode retaining said delayed value within said delayed latch whilst said non-delayed latch is powered down and is susceptible to loss of said non-delayed value; wherein

said delayed latch is formed to have a lower power consumption than said non-delayed latch.

The above, and other objects, features and advantages of this invention will be apparent from the following detailed description of illustrative embodiments which is to be read in connection with the accompanying drawings.

### **BRIEF DESCRIPTION OF THE DRAWINGS**

Figure 1 schematically illustrates a plurality of processing stages to which the present technique is applied;

Figure 2 is a circuit block diagram schematically illustrating a circuit for use in the present technique;

Figure 3 is a circuit diagram schematically illustrating a non-delayed latch and a delayed latch together with an associated comparator and error-recovery logic;

Figures 4A and 4B are a flow diagram schematically illustrating the operation of the circuit of Figure 1;

Figure 5 schematically illustrates a memory circuit including a fast read mechanism and a slow read mechanism;

Figure 6 illustrates an alternative circuit arrangement for a portion of the circuit of Figure 5;

Figure 7 is a flow diagram schematically illustrating the operation of the memory circuit of Figure 5;

Figure 8 illustrates a pipelined bus including non-delayed latches and delayed latches between the bus stages;

Figure 9 is a flow diagram schematically illustrating the operation of the pipelined bus of Figure 8;

Figure 10 schematically illustrates the generation of control signals for controlling a microprocessor that are subject to both non-delayed latching and output and delayed latching and output;

Figure 11 is a flow diagram schematically illustrating one example of the operation of the circuit of Figure 10;

Figure 12 illustrates a processing pipeline including non-delayed latches and delayed latches with those delayed latches being reused as data retention latches during a lower power of operation;

Figure 13 is a flow diagram schematically illustrating the operation of the circuit of Figure 12;



Figure 14 schematically illustrates a plurality of processing stages to which error correction and delayed latches have been applied;

Figure 15 schematically illustrates error correction for data passing through a channel that simply passes the data value unchanged from input to output if no errors occur;

Figure 16 schematically illustrates how error correction is performed for a value-changing logic element such as an adder, multiplier or shifter;

Figure 17 is a flow chart schematically illustrating the operation of the circuit of Figure 14;

Figure 18 schematically illustrates how delayed and non-delayed latches can be used to control the relative phases of clock signals within a processing pipeline;

Figures 19 and 20 schematically illustrate respective uses of stalls and bubble insertion in recovering from errors; and

Figure 21 illustrates a non-delayed and delayed latch for use between processing stages with the delayed latch being reused as a serial scan chain latch.

### **DESCRIPTION OF THE PREFERRED EMBODIMENTS**

Figure 1 illustrates a part of an integrated circuit, which may be a part of a synchronous pipeline within a processor core, such as an ARM processor core produced by ARM limited of Cambridge, England. The synchronous pipeline is formed of a plurality of like processing stages. The first stage comprises processing logic 2 followed by a non-delayed latch 4 in the form of a flip-flop together with a comparator 6 and a delayed latch 8. The term latch used herein encompasses any circuit element operable to store a signal value irrespective of triggering, clock and other requirements. Subsequent processing stages are similarly formed. A non-delayed clock signal 10 drives the processing logic and non-delayed latches 4 within all of the processing stages to operate synchronously as part of a synchronous

pipeline. A delayed clock signal 12 is supplied to the delayed latches 8 of the respective processing stages. The delayed clock signal 12 is a phase shifted version of the non-delayed clock signal 10. The degree of phase shift controls the delay period between the capture of the output of the processing logic 2 by the non-delayed latch 4 and the capture of the output of the processing logic 2 at a later time performed by the delayed latch 8. If the processing logic 2 is operating within limits given the existing non-delayed clock signal frequency, the operating voltage being supplied to the integrated circuit, the body bias voltage, the temperature etc, then the processing logic 2 will have finished its processing operations by the time that the non-delayed latch 4 is triggered to capture its value. Consequently, when the delayed latch 8 later captures the output of the processing logic 2, this will have the same value as the value captured within the non-delayed latch 4. Accordingly, the comparator 6 will detect no change occurring during the delay period and error-recovery operation will not be triggered. Conversely, if the operating parameters for the integrated circuit are such that the processing logic 2 has not completed its processing operation by the time that the non-delayed latch 4 captures its value, then the delayed latch 8 will capture a different value and this will be detected by the comparator 6 thereby forcing an error-recovery operation to be performed. It will be seen that the error-recovery operation could be to replace the output of the non-delayed latch 4 which was being supplied to the following processing stage during the time following its capture with the delayed value stored within the delayed latch 8. This delayed value may additionally be forced to be stored within the non-delayed latch 4 replacing the previously erroneously captured value stored therein.

A meta-stability detector 7 serves to detect meta-stability in the output of the non-delayed latch 4, i.e. not at a clearly defined logic state. If such meta-stability is detected, then this is treated as an error and the value of the delay latch 6 is used instead.

On detection of an error, the whole pipeline may be stalled by gating the non-delayed clock signal 10 for an additional delayed period to give sufficient time for the processing logic in the following processing stage to properly respond to the corrected input signal value being supplied to it. Alternatively, it is possible that upstream processing stages may be stalled with subsequent processing stages being allowed to

continue operation with a bubble inserted into the pipeline in accordance with standard pipeline processing techniques using a counterflow architecture (see the bubble and flush latches of Figure 2). Another alternative is that the entire processing pipeline may be reset with the delayed latch values being forced into the non-delayed latches of each stage and processing resumed. The re-use of the delayed latch value in place of the erroneous value rather than an attempted recalculation ensures that forward progress is made through the processing operations even though an error has occurred.

There are constraints relating to the relationship between the processing time taken by the processing logic within the processing stages and the delay between the non-delayed capture time and the delayed capture time. In particular, the minimum processing time of any processing stage should not be less than the delay in order to ensure that the delayed value captured is not corrupted by new data being outputted from a short delay processing stage. It may be necessary to pad short delay processing stages with extra delay elements to ensure that they do not fall below this minimum processing time. At the other extreme, it needs to be ensured that the maximum processing delay of the processing logic within a processing stage that can occur at any operational point for any operating parameters is not greater than the sum of the normal non-delayed operating clock period and the delay value such that the delay value captured in the delay value latch is ensured to be stable and correct.

There are a number of alternative ways in which the system may be controlled to tune power consumption and performance. According to one arrangement an error counter circuit (not illustrated) is provided to count the number of non-equal detections made by the comparator 6. This count of errors detected and recovered from can be used to control the operating parameters using either hardware implemented or software implemented algorithms. The counter is readable by the software. The best overall performance, whether in terms of maximum speed or lowest power consumption can be achieved by deliberately operating the integrated circuit with parameters that maintain a non-zero level of errors. The gain from operating non-cautious operating parameters in such circumstances exceeds the penalty incurred by the need to recover from errors.

According to an alternative arrangement, a hardware counter is provided as a performance monitoring module and is operable to keep track of useful work and of error recovery work. In particular, the counter keeps count of the number of useful instructions used to progress the processing operations being executed and also keeps  
5 count of the number of instructions and bubbles executed to perform error recovery. The software is operable to read the hardware counter and to use the count values to appropriately balance the overhead of error recovery and its effects on system performance against the reduced power consumption achieved by running the integrated circuit at a non-zero error rate.

10 Figure 2 is a circuit block diagram schematically illustrating a circuit for use in the present technique. The top portion of Figure 2 illustrates circuit elements provided within each processing stage, namely the non-delayed latch 4, the delayed latch 8 and the comparator 6. A meta-stability detector 7 serves to detect meta-  
15 stability in the output of the non-delayed latch 4 and this also triggers generation of an error signal. Error signals from a plurality of such stages are supplied to respective inputs of an OR gate 100 where a global error signal is generated if an error is detected in any processor stage. The global error signal can be used to trigger flush and bubble insertion signals as illustrated. The circuits 102 detect whether the error  
20 signal itself is meta-stable. The error signal is latched with a positively skewed latch, referencing at a higher voltage and a negatively skewed latch, referencing at a lower voltage. If the two disagree in their latched value, this indicates that the error signal was meta-stable and the panic signal is pulled. By latching the error signal and waiting for an entire clock cycle before it sampled (i.e. two latches in series), the  
25 probability of the panic signal being meta-stable is negligible. It is significant that if the panic signal is pulled, then the restored value from the delayed latch could be corrupted due to the meta-stability of the error signal. In this case, the instruction is also invalidated and there is no forward progress. Hence flush the pipeline restart the instruction and lower the clock frequency to ensure that the error signal will not be  
30 meta-stable on the retry of the same instruction (which could otherwise cause an infinite loop of retries).

Figure 3 is a circuit illustrating in more detail the non-delayed latch, the delayed latch, the comparator and at least part of the error-recovery circuitry. The

non-delayed latch 4 can be seen to be in the form of a flip-flop provided by the two latches 14, 16. The delayed latch 8 is in the form of a single feedback element. An XOR gate 18 serves as the comparator. An error signal 20 emerges from the circuit of Figure 3 and may be supplied to the error counter circuit as previously discussed or to other operational parameter adjusting circuits or systems. The error signal 20 serves to switch a multiplexer 22 that forces the delayed value stored within the delayed latch 8 to be stored within the latch 14 of the non-delayed latch 4. meta-stability detecting circuits 24 serve to detect the occurrence of meta-stability within the non-delayed latch 4 and also use this to trigger an error signal which will cause the erroneous meta-stable value to be replaced by the delayed value stored within the delayed latch 8.

Figures 4A and 4B are a flow diagram schematically illustrating the operation of the circuits of Figures 1, 2 and 3.

At step 26 the processing logic from a stage  $i$  produces its output signal at a time  $T_i$ . At step 28 this is captured by the non-delayed latch and forms the non-delayed value. At step 30 the non-delayed value from the non-delayed latch starts to be passed to the following processing stage  $i + 1$  which commences processing based upon this value. This processing may turn out to be erroneous and will need recovering from should an error be detected.

Step 32 allows the processing logic to continue processing for a further time period, the delay time, to produce an output signal at time  $T_i + d$ . This output signal is latched in the delayed latch at step 34. The values within the delayed latch and the non-delayed latch are compared at step 36. If they are equal then no error has occurred and normal processing continues at step 37. If they are not equal, then this indicates that the processing logic at time  $T_i$  had not completed its processing operations when the non-delayed latch captured its value and started to supply that value to the subsequent processing stage  $i + 1$ . Thus, an error condition has arisen and will require correction. At step 38 this correction is started by the forwarding of a pipeline bubble into the pipeline stages following stage  $i$ . At step 40 the preceding stages to stage  $i + 1$  are all stalled. This includes the stage  $i$  at which the error occurred. At step 42, stage  $i + 1$  re-executes its operation using the delayed latch

value as its input. At step 44 the operating parameters of the integrated circuit may be modified as required. As an example, the operating frequency may be reduced, the operating voltage increased, the body biased voltage increased etc. Processing then continues to step 46.

5

If an insufficient number of errors is detected, then the operating parameter controlling circuits and algorithms can deliberately adjust the operating parameters so as to reduce power consumption and to provoke a non-zero error rate.

10        Figure 5 illustrates a memory 100 containing an array of memory cells 102. In this example, a single row of memory cells is illustrated, but as will be familiar to those in this technical field such memory cell arrays are typically large two-dimensional arrays containing many thousands of memory cells. In accordance with normal memory operation, a decoder 104 serves to receive a memory address to be  
15        accessed and to decode this memory address so as to activate one of the word lines 106. The word lines serve to couple the memory cells 102 in that line to respective bit line pairs 108. Depending upon the bit value stored within the memory cell 102 concerned this will induce an electrical change (e.g. a change in voltage and/or a current flow) in the bit lines 108 now coupled to it and the change is sensed by a sense  
20        amplifier 110. The output of the sense amplifier 110 is stored at a first time within a non-delayed latch 112 and subsequently stored at a delayed time within a delayed latch 114. The non-delayed value stored within the non-delayed latch 112 is directly passed out via a multiplexer 116 to a further processing circuit 118 before the delayed value has been stored into the delayed latch 114. When the delayed value has been  
25        captured within the delayed latch 114, a comparator 120 serves to compare the non-delayed value and the delayed value. If these are not equal, then the delayed value is switched by the multiplexer 116 to being the output value from the memory 100 for the particular bit concerned. A suppression signal is also issued from the comparator 120 to the further processing circuit 118 to suppress processing by that further  
30        processing circuit 118 based upon the erroneous non-delayed value which has now been replaced. This suppression in this example takes the form of controlling the clock signal CLK supplied to the further processing circuit 118 to stretch the clock cycle concerned and to delay latching of the new result by that further processing circuit until a time when the delayed value has had a chance to propagate through the

processing circuit concerned to reach the latch at the output of that further processing circuit.

It will be seen that the sense amplifier 110 and the non-delayed latch 112 form  
5 part of the fast read mechanism. The sense amplifier 110 and the delayed latch 114  
form part of the slow read mechanism. In most cases, the fast read result latched  
within the non-delayed latch 112 will be correct and no corrective action is necessary.  
In a small number of cases, the fast read result will differ from the slow read result  
latched within the delayed latch 114 and in this circumstance the slow read result is  
10 considered correct and serves to replace the fast read result with processing based  
upon that fast read result being suppressed. The penalty associated with a relatively  
infrequent need to correct erroneous fast read results is more than compensated for by  
the increased performance (in terms of speed, lower voltage operation, lower energy  
consumption and/or other performance parameters) that is achieved by running the  
15 memory 100 closer to its limiting conditions.

Figure 6 illustrates a variation in part of the circuit of Figure 5. In this  
variation two sense amplifiers 110', 110'' are provided. These different sense  
amplifiers 110', 110'' are formed to have different speeds of operation with one 110'  
20 being relatively fast and less reliable and the other 110'' being relatively slow and  
more reliable. These different characteristics can be achieved by varying parameters  
of the sense amplifier 110', 110'', e.g. construction parameters such as transistor size,  
doping levels, gain etc. A comparator 120' serves to compare the two outputs. The  
output from the fast sense amplifier 110' is normally passed out via the multiplexer  
25 116' prior to the output of the slow sense amplifier 110'' being available. When the  
output of the slow sense amplifier 110'' is available and the comparator 120 detects  
this is not equal to the output of the fast sense amplifier 110', then it controls the  
multiplexer 116' to switch the output value to be that generated by the slow sense  
amplifier 110''. The comparator 120 also triggers generation of a suppression signal  
30 such that downstream processing based upon the erroneous fast read result is  
suppressed.

Figure 7 is a flow diagram illustrating the operation of the circuit of Figure 5.  
At step 122, an address is decoded resulting in respective memory cells being coupled

to their adjacent bit lines using a signal passed by a word line. At step 124, the bit values stored within the selected memory cells and their complements are driven onto the bit line pairs. This causes current flows within the bit lines and voltage changes in the bit lines. The sense amplifiers 110 are responsive to detected currents and/or voltage level changes.

At step 126, the fast data read mechanism samples the value being output from the memory cell at that time. At step 128 this fast read data value is passed to subsequent processing circuits for further processing upon the assumption that it is correct. At step 130, the slow data reading mechanism samples a slow read data value. Step 132 compares the fast read value and the slow read value. If these are the same, then normal processing continues at step 134. However, if the sampled values are different, then step 136 serves to issue a suppression signal to the further circuits to which the fast read value has been passed and also to issue the slow read value in place of the fast read value to those further circuits such that corrective processing may take place.

Figure 8 illustrates the use of the present techniques within a pipelined bus 140. The pipelined bus 140 contains a number of latches 142 which serve to store data values being passed along the bus. As an example of such a pipelined bus 140 there is known the AXI buses designed by ARM Limited of Cambridge, England. In this arrangement the destination for the data value being passed along the pipelined bus 140 is a digital signal processing circuit 144. This digital signal processing (DSP) circuit 144 does not in itself implement the non-delayed latching and delayed latching techniques discussed previously. In alternative arrangements the destination for the data value being passed along the pipelined bus could be a device other than a DSP circuit, for example, a standard ARM processor core that does not itself implement the delayed and non-delayed latching techniques.

Associated with each of the non-delayed latches 142 is a respective delayed latch 146. These delayed latches 146 serve to sample the signal value on the bus at a time later than when this was sampled and latched by the non-delayed latch 142 to which they correspond. Thus, a delay in the data value being passed along the bus for whatever reason (e.g. too low an operational voltage being used, the clock speed



being too high, coupling effects from adjacent data values, etc) will result in the possibility of a difference occurring between the values stored within the non-delayed latch 142 and the delayed latch 146. The final stage on the pipeline bus 140 is illustrated as including a comparator 147 which compares the non-delayed value and the delayed value. If these are not equal, then the delayed value is used to replace the non-delayed value and the processing based upon the non-delayed value is suppressed such that the correction can take effect (the bus clock cycle may be stretched). It will be appreciated that these comparator and multiplexing circuit elements will be provided at each of the latch stages along the pipeline bus 140, but these have been omitted for the sake of clarity from Figure 8.

As the DSP circuit 144 does not itself support the non-delayed and delayed latching mechanism with its associated correction possibilities, it is important that the data value which is supplied to the DSP circuit 144 has been subject to any necessary correction. For this reason, an additional buffering latch stage 148 is provided at the end of the pipelined bus 140 such that any correction required to the data value being supplied to that latch and the attached DSP circuit 144 can be performed before that data value is acted upon by the DSP circuit 144. The buffering latch 148 can be placed in sufficient proximity to the DSP circuit 144 that there will be no issue of an insufficient available propagation time etc. causing an error in the data value being passed from the buffering latch 148 to the DSP circuit 144.

It will be appreciated that the bus connections between the respective non-delayed latches 142 can be considered to be a form of processing logic that merely passes the data unaltered. In this way, the equivalence between the pipelined bus embodiment of Figure 8 and the previously described embodiments (e.g. Figure 1) will be apparent to those familiar with this technical field.

Figure 9 is a flow diagram illustrating the operation of Figure 8. At stage 150 a non-delayed signal value is captured from the bus line. At step 152 the non-delayed value is then passed to the next bus pipeline stage. At step 154 the corresponding delayed latch 146 captures a delayed bus signal. At step 156 the comparator 147 compares the delayed value with the non-delayed value. If these are equal, then normal processing continues at step 158. If the two compared values are not equal,

then step 160 serves to delay the bus clock and replace the non-delayed value with the delayed value using the multiplexer shown in Figure 8.

Figure 10 illustrates a further example embodiment using the present techniques. In this example embodiment an instruction from an instruction register within a processor core is latched within an instruction latch 162. From this instruction latch 162, the instruction is passed to a decoder 164 which includes a microcoded ROM serving to generate an appropriate collection of processor control signals for storage in a non-delayed control signal latch 166 and subsequent use to control the processing performed by the processor core in accordance with the instruction latched within the instruction latch 162. The control signals output from the decoder 164 are also latched within a delayed control signal latch 168 at a later time to when they were latched within the non-delayed control signal latch 166. The delayed control signal values and the non-delayed control signal values can then be compared. If these are not equal, then this indicates that corrective action is necessary. A suppression operation is triggered by the detection of such a difference and serves to stop subsequent processing based upon the inappropriate latch control signal values. It may be that in some circumstances the only effective recovery option is to reset the processor as a whole. This may be acceptable. In other situations, the error in the control signals might be such that a less drastic suppression and recovery mechanism is possible. As an example, the particular erroneous control signal may not yet have been acted upon, e.g. in the case of a multi-cycle program instruction where some processing operations do not commence until late in the overall execution of the multi-cycle instruction. An example of this is a multiply-accumulate operation in which the multiply portion takes several clock cycles before the final accumulate takes place. If there is an error in the control signal associated with the accumulate and in practice an accumulate is not required, but merely a pure multiply, then it would be possible to suppress the accumulate by correcting the control signal being applied to the accumulator before the adder had sought to perform the accumulate operation.

Figure 11 illustrates one example of the operation of the circuit of Figure 10. At step 170, a multiply-accumulate control signal is read from the decoder 164 (microcoded ROM). At step 172, this multiply-accumulate control signal is latched

within the non-delayed control signal latch 166 and output to the various processing elements within the processor core. At step 174, the multiply operands are read from the register file and the multiply operation is initiated. At step 176, the control signals output by the instruction decoder 164 are re-sampled by the delayed control signal latch 168. At step 178, the non-delayed control signals and the delayed control signals are compared. If these are equal, then normal processing continues at step 180. However, if these are not equal, then processing proceeds to step 182 where a determination is made as to whether the multiply operation has yet completed. If the multiply operation has completed, then the erroneous accumulate operation will have started and the best option for recovery is to reset the system as a whole at step 184. However, if the multiply operation is still in progress, then step 186 can be used to reset the adder and cancel the accumulate operation with the desired multiply operation output result being generated at step 188, as was originally intended by the program instruction stored within the instruction latch 162.

Figure 12 illustrates a modification of the circuit illustrated in Figure 1. In this embodiment the delayed latches 190 serve the additional function of data retention (balloon) latches for use during a standby/sleep mode of operation (low power consumption mode). The function of the delayed latches 190 during normal processing operations is as previously described. However, when a sleep controller 192 serves to initiate entry into a low power consumption mode of operation it stops the non-delayed clock and the delayed clock such that the delayed latches 190 are all storing data values corresponding to their respective non-delayed latches. At this point, the voltage supply to the non-delayed latches and the associated processing circuits is removed such that they are powered down and lose their state. However, the voltage supplied to the non-delayed latches 190 is maintained such that they serve to retain the state of the processing circuit concerned. When the system exits from the low power consumption mode, the processing logic and the non-delayed latches are powered up again when the comparator detects a difference in the values in the non-delayed latch and the delayed latch 190 it triggers replacement of the erroneous value within the non-delayed latch with the correct value held within the associated delayed latch 190. It will be appreciated that since the delayed latches 190 are subject to less stringent timing requirements than their non-delayed counterparts they can be formed in a way where they may have a lower speed of operation but be better suited to low

power consumption during the low power consumption mode (e.g. high threshold voltages resulting in slower switching but with a reduced leakage current). In this way, the error correcting delayed latches which are used during normal processing can be reused during the low power consumption mode as data retention latches thereby  
5 advantageously reducing the overall gate count of the circuit concerned.

Figure 13 is a flow diagram schematically illustrating the operation of the circuit of Figure 12. At step 194, the integrated circuit is in its normal operational processing mode. At step 196, the processing logic stage produces an output signal at  
10 a non-delayed time. At step 198, the non-delayed latch captures that output signal. At step 200 the non-delayed signal within the non-delayed latch is passed to the next processing stage. At step 202, the output from the processing stage at a delayed time is generated and is available for capture by the delayed latch. At step 204, the integrated circuit is triggered to adopt a low power consumption mode and the speed  
15 controller 192 serves to initiate the power down of the processing circuits while maintaining the power to the delayed latches 190. At step 206, the delayed latch 190 captures the delayed signal value. It may be that the capture of the delayed signal value by the delayed latch at step 206 takes place before the switch to the low power mode at step 204. At step 208, the non-delayed latch is powered down and its stored  
20 value is lost. The integrated circuit can remain in this state for a long period of time. When desired, step 210 triggers the sleep controller 192 to exit the low power consumption mode and revert back to the operational mode. At step 212, power is restored to the non-delayed latches and the associated processing logic with the delayed data values within the delayed latches 190 being used to repopulate the  
25 pipeline stages as necessary to restore the system to its condition prior to the low power consumption mode being entered.

Figure 14 schematically illustrates a plurality of processing stages to which error correction control and delayed latches have been applied. The processing stages  
30 form part of an integrated circuit that may be part of a synchronous pipeline within a processor core, part of a communication bus or part of a memory system. The first processing stage comprises either a channel for communication of data or processing logic 1014, a non-delayed latch 1016, a delayed latch 1018, a comparator 1024 that compares outputs of the delayed latch and the non-delayed latch and outputs a control

signal to a multiplexer 1020 determining whether the delayed signal value or the non-delayed signal value is supplied as input to a subsequent processing stage or channel 1016. The channel/logic 1014 and the non-delayed latch 1016 are driven by a non-delayed clock signal whereas the delayed latch 1019 is driven by a delayed clock  
5 signal which is a phase-shifted version of the non-delayed clock signal.

If the comparator 1024 detects a difference between the non-delayed signal value and the delayed signal value this indicates that either the processing operation was incomplete at the non-delayed capture time in the case that element 1014  
10 represents processing logic or that the signal from the previous pipeline stage had not yet reached the present stage in the case of the element 1014 representing a data channel. In the event that such a difference is in fact detected, the value stored in the delayed latch 1018 is the more reliable data value since it was captured later when the processing operation is more likely to have been completed or the data from the  
15 previous stage is more likely to have arrived via the data channel. By supplying the result from the delayed latch to the next processing stage 1030 and suppressing use of the non-delayed value in subsequent processing stages, forward progress of the computation can be ensured. However, the reliability of the delayed signal value stored in the delayed latch 1018 can be compromised in the event that a single event  
20 upset occurred and corrupted the delayed value. The single event upset is effectively a pulse so it may well be missed by the non-delayed latch but picked up by the delayed latch. Such a single event upset will result in the comparator detecting a difference between the delayed and non-delayed values as a direct result of the single event upset and will then propagate the corrupted delayed value to subsequent  
25 processing stages. A single event upset that corrupts the non-delayed value will not be problematic since it will result in suppressing use of the erroneous non-delayed value and propagating the delayed value to subsequent stages.

The arrangement of Figure 14 reduces the likelihood of a corrupted delayed  
30 value progressing through the computation by providing a cross-check of data integrity by provision of an error detection module 1026, an error correction module 1028 and a multiplexer 1022 that is controlled by the error detection module 1026 to supply either the delayed value from the delayed latch directly to the comparator 1024 or alternatively to supply an error corrected value output by the error correction

module 1028. Upstream of the channel/logic unit 1014 a data payload of eight bits is error correction encoded and four redundancy bits are added to the data payload to form a twelve-bit signal. The twelve-bit signal passes through the channel/logic unit 1014 and its value is captured by both the non-delayed latch 1016 and the delayed latch 1018. However, a delayed value of the signal derived from the delayed latch 1018 is also supplied as input to the error detection module 1026, which determines from the 12-bit error-correction encoded signal whether any errors have occurred that affect the delayed value. In an alternative arrangement a further latch could be provided to supply a signal value to the error detection module 1018, that captures the signal value at a time slightly later than the delayed latch 1018. The error-checking must be performed on a value captured at the same time as the delayed value is captured or slightly later to ensure that any random error that occurred between capture of the non-delayed value and capture of the delayed value is detected.

A given error correction code is capable of detecting a predetermined number of errors and of correcting a given number of errors. Thus the error detection module 1026 detects whether any errors have occurred and, if so, if the number of errors is sufficiently small such that they are all correctable. If correctable errors are detected then the signal value is supplied to the error correction module 1028 where the errors are corrected using the error correction code and the corrected delayed value is supplied to the comparator 1024. If it is determined by the comparator 1024 that the corrected delayed value differs from the non-delayed value then the error recovery procedure is invoked so that further propagation of the non-delayed value is suppressed in subsequent processing stages and the operations are instead performed using the corrected delayed value. On the other hand, if the comparator 1024 determines that the corrected delayed value is the same as the delayed value then there are two alternative possibilities for progressing the calculation. Firstly, the error recovery mechanism could nevertheless be invoked so that the non-delayed value is suppressed in subsequent processing stages and replaced by the corrected delayed value. Alternatively, since the non-delayed value is determined to have been correct (as evidenced by the equality of the non-delayed value and the corrected delayed value), the error recovery mechanism could be suppressed (despite the detection of an error in the delayed value) thus allowing the non-delayed value to continue to progress through the subsequent processing stages. However, if uncorrectable errors

are detected in the delayed value by the error detection module 1026 then a control signal is supplied to suppress use of the corrupted delayed value. In this case forward progress of the computation cannot be achieved. The type of error correction encoding applied differs according to the nature of the channel/processing logic 1014.

Processing logic can be categorised as either value-passing or value-altering. Examples of processing logic that is value-passing are memory, registers and multiplexers. Examples of value-altering processing logic elements are adders, multipliers and shifters. Error detection and correction for value-altering processing logic elements is more complex than for value-passing processing logic elements because even when no error has occurred the value output by the logic stage 1014 is likely to be different from the input twelve-bit signal 1013.

Figure 15 schematically illustrates error correction for data passing through a channel that simply passes the data value unchanged from input to output if no errors occur. In the case of such value-passing processing logic it is convenient to use a linear block code such as a Hamming code for error correction and detection. Linear block codes typically involve forming a codeword in which the original data payload bits remain in the codeword unchanged but some parity bits (or redundancy bits) are added. Hamming codes are simple single-bit error correction codes and for an (N, K) code, N is the total number of bits in the codeword and K is the number of data bits to be encoded. The presence and location of an error is detected by performing a number of parity checks on the output codeword. The Hamming code comprises N-K parity bits, each of which is calculated from a different combination of bits in the data. Hamming codes are capable of correcting one error or detecting two errors. The number of parity bits (or redundancy bits required is given by the Hamming rule  $K+p+1 \leq 2^p$ , where p is the number of parity bits and  $N=K+p$ .

As illustrated in Figure 15 input to the channel is a 12 bit codeword comprising eight data bits and four parity or redundancy bits. Parity checks are performed by an error detection/correction module 1116 on the output from the channel 1114. Any single-bit error in the 12-bit codeword is detected and corrected prior to output of the codeword by the error detection/correction module 1116. If detected errors are uncorrectable the error detection/correction module 1116 outputs a

signal indicating that this is the case. Although simple codes such as Hamming codes have been described in relation to Figure 11 for use with value-passing processing logic, it will be appreciated that other error correction codes such as convolutional codes could alternatively be used.

5

Figure 16 schematically illustrates how error correction is performed for a value-changing logic element such as an adder, multiplier or shifter. In the case of value-altering processing logic arithmetic codes such as AN codes, residue codes, inverse residue codes or residue number codes may be used to detect and correct  
10 random errors in the output of the processing logic.

Arithmetic codes can be used to check arithmetic operators. Where  $\otimes$  represents the operator to be checked the following relation must be satisfied:

15 
$$\text{Code}(X \otimes Y) = \text{code } X \otimes \text{code } Y$$

AN codes are arithmetic codes that involve multiplying the data word by a constant factor, for example a 3N code can be used to check the validity of an addition operation by performing the following comparison:

20

$$\begin{aligned} 3N(X) + 3N(Y) &?= 3N(X+Y) \\ 3X + 3Y &?= 3(X+Y). \end{aligned}$$

A further example of a class of arithmetic codes are residue codes, in which a  
25 residue (remainder of division by a constant) is added to the data bits as check bits e.g. a 3R code involves modulo (MOD) 3 operations and the following check is applied:

$$X \text{ MOD } 3 + Y \text{ MOD } 3 ?= (X+Y) \text{ MOD } 3$$

30

Consider the numerical example of  $X=14$  and  $Y=7$ :

$14 \text{ MOD } 3 = 2$  (codeword 111010, with last two bits as residue);

$7 \text{ MOD } 3 = 1$  (codeword 011101);



$X+Y = 21$  (10101);  
and  $21 \text{ MOD } 3 = 0$ ;  
sum of residues  $\text{MOD } 3 = (2 + 1) \text{ MOD } 3 = 0 = \text{residue of } (X+Y)$ .

5        Figure 16 schematically illustrates use of a 7R arithmetic code for checking of  
an addition operation in the channel/logic units 1014 of Figure 10. The addition  
operation to be checked is  $X + Y$ , where  $X$  and  $Y$  are eight-bit data words. Each data  
word has a four check bits having values  $X \text{ MOD } 7$  and  $Y \text{ MOD } 7$  respectively.  $X$   
 $\text{MOD } 7$  and  $Y \text{ MOD } 7$  are supplied as operands to a first adder 1210 and the output of  
10 this adder is supplied to logic that determines the value  $(X \text{ MOD } 7 + Y \text{ MOD } 7)$   
 $\text{MOD } 7$  and supplies the result as a first input to a comparator 1250. A second adder  
1230 performs the addition  $(X + Y)$ , supplies the result to a logic unit 1240 that  
calculates  $(X+Y) \text{ MOD } 7$  and supplies the result as a second input to the comparator  
1250. If the comparator detects any difference between the two input values then an  
15 error has occurred.

Figure 17 is a flow chart that schematically illustrates the operation of the  
circuit of Figure 14 that comprises error correction control of the delayed latch value.  
At stage 1310 a twelve-bit error correction encoded signal value is input to the  
20 channel/logic unit 1014. Next, at stage 1320, the non-delayed latch 1016 captures the  
output from the channel/logic unit 1014 at time  $T_i$  and the captured value is forwarded  
to subsequent processing logic stage  $I+1$  at stage 1330. At stage 1340 the delayed  
latch 1018 captures the output signal at time  $T_i+d$ . At stage 1350, the error detection  
logic captures the output from the channel/logic unit 1014 at time  $T_i+(d + \delta)$ .  
25 Although  $\delta$  in preferred arrangements  $\delta$  is zero so that value output by the delayed  
value itself is actually error checked, the output may alternatively be captured a short  
after the delayed latch captures the output signal at  $T_i+d$ . The capture of the value for  
supply to the error detection circuit is appropriately timed to ensure that any random  
error in the delayed value is detected. At stage 1360, the error detection module 1026  
30 determines whether the delayed output signal has an error using the redundancy bits.  
If an error is detected it is then determined whether the error is correctable at stage  
1370, which will depend on how many bits are affected. For example, a Hamming  
code can only correct a single bit error. If it is determined at stage 1370 that the error

is correctable then the process proceeds to stage 1390, whereupon the error is corrected and the corrected delayed value is selected at the multiplexer 1022 and supplied to the comparator 1024. However, if it is determined at stage 1370 that detected errors are not correctable then a control signal is generated indicating that an uncorrectable error has occurred. In this case forward progress of the computation cannot be reliably performed. At stage 1392 the comparator 1024 determines whether the error-checked delayed value is equal to the non-delayed value and if so forward progress of the computation continues. Otherwise the process to the sequence of steps described in relation to Figure 4B, involving suppression of the non-delayed value and its replacement by the delayed value in subsequent processing stages is carried out.

Figure 18 illustrates the use of the present technique to dynamically adjust the relative timing between processing stages. It is known that in a pipelined processing environment, the processing stages may take different times to complete their respective operations. Ideally the processing stages would all be balanced to take the same time and for their respective times to vary in the same way with changes in surrounding conditions. However, this is not practical in many cases and it may be that a collection of processing stages that are balanced at one operational voltage or temperature are not balanced at another operational voltage or temperature. Furthermore, manufacturing variation and other characteristics may result in considerable differences between processing stage timings which upsets the designed balance therebetween. In these cases, the clock frequency and other operational parameters are chosen with respect to a worst-case scenario such that the processing stages will be sufficiently closely balanced so as to be operational under all conditions.

The present technique allows a more selective and indeed dynamic approach to be taken. A pipelined processing circuit 2000 includes delayed latches 2002 which can be used to detect the occurrence of errors in the signal values being captured by the non-delayed latches. The occurrence of these errors is fed back to a clock phase control circuit 204 which serves to adjust the relative phases of the clock signals being supplied to respective latches within the main path, i.e. the non-delayed latches. In this way, an adjustment is made whereby time is effectively borrowed from one processing stage and allocated to another processing stage. This may be achieved by

tapping the clock signals to be used by the respective non-delayed latches from selectable positions within a delay line along which the basic clock signal is propagated.

5           The illustrated example, the processing logic between latch  $L_A$  and latch  $L_B$  is slower in operation than the processing logic in the subsequent stage. Accordingly, the clock signal being supplied to the non-delayed latch  $L_B$  can be phase shifted so as to delay the rising edge of that clock signal (assuming rising edge latch capture) and thereby to extend the time available for the slow processing logic. This reduces the  
10   time available for the processing logic within the subsequent processing stage assuming that this is operating on the same basic clock signal as the other stage elements excluding the latch  $L_B$ .

          This timing balancing between processing stages can be performed  
15   dynamically during the ongoing operation of the circuit using feedback from the errors in operation detected using the delay latches. Alternatively, the balancing can be performed as a one-off operation during a manufacturing test stage or during a “golden boot” of the integrated circuit. The delayed latches shown in Figure 18 are used for the purpose of timing balancing between processing stages and can thereafter  
20   be used for the control of operating parameters and error correction as discussed above, e.g. in relation to Figure 1. In this way, the provision of the delayed latches is further used to also control relative clock timings.

          Figure 19 illustrates a simple approach to pipeline error recovery based on  
25   global clock gating. In the event that any stage detects an error, the entire pipeline is stalled for one cycle by gating the next global clock edge. The additional clock period allows every stage to recompute its result using the delayed latch as input. Consequently, any previously forwarded errant values will be replaced with the correct value from the delayed latch. Since all stages re-evaluate their result with the  
30   delayed latch input, any number of errors can be tolerated in a single cycle and forward progress is guaranteed. If all stages produce an error each cycle, the pipeline will continue to run, but at  $\frac{1}{2}$  the normal speed.

It is important that errant pipeline results not be written to architected state before it has been validated by the comparator. Since validation of delayed values takes two additional cycles (i.e., one for error detection and one for panic detection), there must be two non-speculative stages between the last delayed latch and the writeback (WB) stage. In our design, memory accesses to the data cache are non-speculative, hence, only one additional stage labelled ST for stabilise is required before writeback (WB). The ST stage introduces an additional level of register bypass. Since store instructions must execute non-speculatively, they are performed in the WB stage of the pipeline.

Figure 19 gives a pipeline timing diagram of a pipeline recovery for an instruction that fails in the EX stage of the pipeline. The first failed stage computation occurs in the 4<sup>th</sup> cycle, but only after the MEM stage has computed an incorrect result using the errant value forward from the EX stage. After the error is detected, a global clock stall occurs in the 6<sup>th</sup> cycle, permitting the correct EX result in the Razor shadow latch to be evaluated by the MEM stage. IN the 7<sup>th</sup> cycle, normal pipeline operation resumes.

In aggressively clocked designs, it may not be possible to implement global  
clock gating without significantly impacting processor cycle time. Consequently, a fully pipelined error recover mechanism based on counterflow, pipelining techniques has been implemented. The approach, illustrated in Figure 20, places negligible timing constraints on the baseline pipeline design at the expense of extending pipeline recovery over a few cycles. When a non-delayed value error is detected, two specific actions must be taken. First, the errant stage computation following the failing non-delayed latch must be nullified. This action is accomplished using the bubble signal, which indicates to the next and subsequent stages that the pipeline slot is empty. Second, the flush train is triggered by asserting the stage ID of failing stage. In the following cycle, the correct value from the delayed latch data is injected back into the pipeline, allowing the errant instruction to continue with its correct inputs. Additionally, there is a counterflow pipeline whereby the flush train begins propagating the ID of the failing stage in the opposite direction of instructions. At each stage visited by the active flush train, the corresponding pipeline stage and the one immediately preceding are replaced with a bubble. (Two stages must be nullified

to account for the twice relative speed of the main pipeline.) When the flush ID reaches the start of the pipeline, the flush control logic restarts the pipeline at the instruction following the errant instruction. In the event that multiple stages experience errors in the same cycle, all will initiate recovery but only the non-delayed error closest to writeback (WB) will complete. Earlier recoveries will be flushed by later ones. Note that the counterflow pipeline may not be the same length as the forward pipeline so that, for example, the flush train of the counterflow pipeline could be two pipeline stages deep whereas the forward pipeline may be twelve stages deep.

Figure 20 shows a pipeline timing diagram of a pipelined recovery for an instruction that fails in the EX stage. As in the previous example, the first failed stage computation occurs in the 4<sup>th</sup> cycle, when the second instruction computes an incorrect result in the EX stage of the pipeline. This error is detected in the 5<sup>th</sup> cycle, causing a bubble to be propagated out of the MEM stage and initiation of the flush train. The instruction in the EX, ID and IF stages are flushed in the 6<sup>th</sup>, 7<sup>th</sup> and 8<sup>th</sup> cycles, respectively. Finally, the pipeline is restarted after the errant instruction in cycle 9, after which normal pipeline operation resumes.

Recall from the description of Figure 2 above, that in the event that circuits 102 detect meta-stability in the error signal then a panic signal is asserted. In this case, the current instruction (rather than the next instruction) should be re-executed. When such a panic signal is asserted, all pipeline state is flushed and the pipeline is restarted immediately after the least instruction writeback. Panic situations complicate the guarantee of forward progress, as the delay in detecting the situation may result in the correct result being overwritten in the delayed latch. Consequently, after experiencing a panic, the supply voltage is reset to a known-safe operating level, and the pipeline is restarted. One re-tuned, the errant instruction should complete without errors as long as returning is prohibited until after this instruction completes.

A key requirement of the pipeline recover control is that it not fail under even the worst operating conditions (e.g. low voltage, high temperature and high process variation). This requirement is met through a conservative design approach that validates the timing of the error recovery circuits at the worst-case subcritical voltage.

Figure 21 schematically illustrates the re-use of a delayed latch 2100 as a serial scan chain latch. This is achieved by the provision of a multiplexer 2102 controlled by the scan enable signals which allow a serial scan data value to be written into the delay latch or serially read from the delayed latch as required. Furthermore, the normal mechanism which allows the delayed latch value to replace the non-delayed latch value is exploited to allow a serial scan chain value to be inserted into the operational path.

Although illustrative embodiments of the invention have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to those precise embodiments, and that various changes and modifications can be effected therein by one skilled in the art without departing from the scope and spirit of the invention as defined by the appended claims.